REMARKS

[0001] Claims 1-20 are pending in the case. In the Office Action, Claims 15-20 were rejected under 35 U.S.C. §102(e) in view of U.S. Patent No. 6,269,402 Lin et al (hereinafter Lin). Claims 1-14 were rejected under 35 U.S.C. §103(a) as unpatentable in view of Lin and U.S. Patent No. 6,877,036 to Smith et al., (hereinafter Smith). The Office Action includes a request for information under 37 CFR §1.105. In response to this request, Applicants have included an excerpt from the source "Open Transaction Manager Access Guide and Reference." The excerpt is included with this paper.

[0002] Applicants amended Claim 15 to resolve two typographical errors. Applicants also amended two paragraphs in the specification to correct incorrect acronyms. No new matter was added.

[0003] Applicants note that the Office Action mailed November 11, 2005 fails to discuss whether the proposed amendments to Claims 8 and 15 have been entered. Since prosecution has been reopened, Applicants presume the proposed amendments have been entered. The pending claims are believed to be in condition for allowance, and Applicants respectfully request the prompt allowance of Claims 1-20.

REJECTION OF CLAIMS 15-20 UNDER 35 U.S.C. §102(e)

[0004] The Office Action rejected Claims 15-20 under 35 U.S.C. §102(e) in view of Lin. Applicants respectfully traverse this rejection.

[0005] It is well settled that under 35 U.S.C. §102 "an invention is anticipated if . . . all the claim limitations [are] shown in a single art prior art reference. Every element of the claimed invention must be literally present, arranged as in the claim. The identical invention must be shown in as complete detail as is contained in the patent claim." *Richardson v. Suzuki Motor Co., Ltd.*, 9 U.S.P.Q.2d 1913, 1920 (Fed. Cir. 1989). Applicants respectfully assert that Lin does not teach or disclose each element of the independent claims.

[0006] Claim 15 recites, in pertinent part:

> a request module configured to receive a client request;
> a response generator which receives the client request from the request module and generates an appropriate response;
> an **unpaired message module** which **analyzes** the response message generated by the response generator and configured to **distinguish a paired message from an unpaired message** in response to a communication disruption between the client and the server and to **store paired** messages in a **paired response data structure** and **unpaired** messages in an **unpaired response data structure**, the at least one unpaired message comprising a communication response for a specific client; and
> a response module which communicates paired and unpaired messages to a client.

(Claim 15 emphasis added).

In particular, Claim 15 recites an unpaired message module that **analyzes** the response message to **distinguish** paired messages from unpaired messages and **stores** paired messages **and** unpaired messages in **separate** data structures. In other words, the upaired message module performs three distinct operations: analyzing, distinguishing, and storing.

[0007] The Office Action suggests that most of these elements are "literally present, [and] arranged as in the claim" in Lin in col. 3, lines 5-27. The Office Action suggests that the response module is "literally present" in Lin in col. 5, lines 35-63. Applicants respectfully disagree.

[0008] The present application has been published as US publication No. 2002/0133563 (hereinafter '563). The present invention serves to facilitate automatic detection, storage, and delivery of unpaired messages in the event of a communication disruption. The present invention provides for well known network communication protocols which allow a client to send a request and the server to send a response. See '563 ¶¶14-15. As explained in '563, an unpaired message is a response that was not deliverable due to a communication disruption. See '563 ¶18.

[0009] Because responses are queued, unpaired messages can cause problems. See '563 ¶¶19-20. The present invention adds logic to the server to overcome these problems. This logic is contained, at least in part, in the unpaired message module 110.

See '563 ¶43, Fig. 1. The features and functionality of the unpaired message module 110 recited in Claim 15 are described in '563 in ¶¶ 43-46. Specifically, the unpaired message module 110 analyzes response messages. See '563, ¶¶ 43-44. The unpaired message module 110 distinguishes between paired messages and unpaired messages. See '563, ¶ 43. The unpaired message module 110 then stores the paired messages in one data structure, a paired response data structure, and stores unpaired messages in a second data structure, an unpaired response data structure. See '563, ¶¶ 44-45.

[0010] In stark contrast, Lin in general, and in the sections relied upon in the Office Action, fail to teach or disclose an unpaired message module with the limitations recited in Claim 15. Lin, in general, relates to managing of a communication disruption between a client and a server. See Lin Abstract. When communication is disrupted, the client and server each set up a session transition control block. See Lin Abstract. The session transition control block is used after a communication disruption to send new packets with the second envelope identifier. Lin col. 6, lines 37-40. The session transition control block is also then used to route messages stored during the interruption that have the first envelop identifier. Lin col. 6, lines 40-44. Resumed communication may use a different bearer network and/or different network parameters. Lin col. 6, lines 20-23. Hence, the identical session transition control blocks at the server and client serve as cross-reference routing tables once communication resumes. The session transition control blocks do not store unpaired messages as suggested in the Office Action. See Office Action page 2-3.

[0011] Lin clearly teaches how communication disruption between a **single** client and **single** server affects both the **single** client and the **single** server. When communication is disrupted, an undeliverable response at the client or server becomes an unpaired message for that particular client/server. However, the unpaired message module deals with both paired and unpaired messages. The unpaired messages correspond to a single client while the paired messages relate to responses for clients **(plural)** having no communication disruption. See '563 ¶18. This means that the unpaired message module distinguishes between paired message and unpaired messages.

[0012] The present invention handles both normal messages to connected clients (connected clients means there is not communication disruption) and unpaired messages

to one or more unconnected clients. Lin is silent regarding paired messages. Consequently, Lin fails to teach or disclose **analyzing** the response message to **distinguish** paired messages from unpaired messages and **storing** paired messages **and** unpaired messages in **separate** data structures.

[0013] Furthermore, Lin clearly teaches that both the server and the client include the logic to set up and manage a session transition control block. Lin col. 3, lines 19-21, col. 5, lines 64-66. This means that the same logic must be executing on the client as is on the server. Consequently, the software of the client is more complicated and larger than if the client did not have to maintain/manage a session transition control block.

[0014] Applicants submit that this duplicate logic teaches away from the benefits and advantages of the present invention. In Claim 15, the preamble recites that the "server compis[es]...an unpaired message module." Therefore, the unpaired message module resides on the server. This means that that client need not include any logic for handling unpaired messages. Consequently, the present invention enables use of "thin" clients whose benefits and advantages are explained in '563 ¶¶13, 22.

[0015] Therefore, Applicants submit that "[e]very element of the claimed invention" is not "literally present" in Lin. *Richardson v. Suzuki Motor Co., Ltd.*, 9 U.S.P.Q.2d 1913, 1920 (Fed. Cir. 1989). Lin fails to teach an unpaired message module configured to analyze response messages for both connected clients and unconnected clients, distinguish paired messages from unpaired messages, and store the paired messages in a paired response data structure and unpaired messages in an unpaired response data structure.

Claim 18

[0016] Regarding Claim 18, the Office Action rejected Claim 18 by relying on Lin col. 5, lines 35-63. Claim 18 relates to the server sending unpaired messages in response to a request for the unpaired messages from the client. Specifically, Claim 18 recites "the response module is configured to send all unpaired messages stored in the unpaired response data structure in response to a request from the client." This means that the unpaired messages are not sent until the client requests the unpaired messages.

[0017] At col. 5, lines 35-63, Lin teaches the main method of operation for the client and server to manage a communication disruption, as explained above. However, Lin fails to teach or describe the client sending a particular request for the unpaired messages as recited in Claim 18. Therefore, Lin fails to teach each element of Claim 18.

Claims 16, 17, 19, and 20

[0018] Claims 16, 17, 19, and 20 depend from Claim 15 and are therefore allowable for at least the same reasons as Claim 15 explained above. Therefore, Applicants submit that Claims 16, 17, 19, and 20 are also allowable over Lin.

REJECTION OF CLAIMS UNDER 35 U.S.C. §103(a)

[0019] The Office Action rejected Claims 1-14 under 35 U.S.C. §103(a) as obvious in view of Lin in view of Smith. Applicants respectfully traverse this rejection.

[0020] Applicants respectfully assert that neither Lin nor Smith teach or suggest all of the elements of the rejected claims in view of the previous remarks. To establish obviousness, the combination of the prior art references must teach or suggest all the claim limitations. See MPEP § 2142.

[0021] Applicants respectfully assert that Lin and Smith fail to teach or suggest all the claim limitations of the independent Claims 1 and 8. Specifically, the references fail to teach or disclose a server that detects and stores an unpaired message and distinguishes unpaired messages from paired messages in response to a communication disruption. The Office Action relies on Lin to argue that these limitations are obvious. Lin's failure to teach these limitations is explained above. Claim 1 recites that a protocol is utilized that "allows the client to request at least one unpaired message." While Col. 3, lines 5-27 describes how communication may resume, Lin fails to teach that the unpaired messages are sent in response to a request for unpaired messages from the client, as discussed in relation to Claim 18 above.

[0022] The Office Action relies on Smith for the teaching that the unpaired message data structure is an unpaired message queue. Smith teaches an adapter card configured to handle connection management burdens for a server. See Smith Abstract. At col. 8, line 65-col. 9 line 26 Smith teaches filling an output queue as needed.

[0023] In addition, "it is insufficient that the prior art disclose[] the components of the patented device, either separately or used in other combinations; there must be some teaching, suggestion, or incentive to make the combination made by the inventor." *Northern Telecom, Inc. v. Datapoint Corp.*, 908 F.2d 931, 934 (Fed. Cir. 1990). Applicants submit that one of skill in the art would not find motivation in Lin or Smith to combine the two because the combination would still lack a server that fails to detect and store an unpaired message and distinguish unpaired messages from paired messages in response to a communication disruption.

[0024] Therefore, Applicants submit that because Lin fails to include the novel elements that are part of independent Claims 1 and 8 and Smith fails to teach or suggest the missing elements, Claims 1 and 8 are nonobvious in view of Lin and Smith. Claims 2-7 and 9-14 are nonobvious due to the dependency and/or remarks presented above. Applicants request that the rejection of Claims 1-14 be withdrawn.

Request for Information

[0025] The Office Action requested information on the Open Transaction Message Access Protocol identified on page 6, lines 9-17 of the specification. In response, Applicants have included pages 1-10 of the manual "Open Transaction Manager Access Guide and Reference." The whole manual is available at the web link http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims9.doc.pdf/dfsotmg2.pdf?noframes=true. Based on the manual, Applicants have requested that the acronym OTMA on page 6 lines 14 be amended to replace "Message" with "Manager."

Conclusion

[0026] Therefore, Lin fails to teach or disclose all the elements of Claim 15. Specifically, Lin fails to teach or disclose an unpaired message module that **analyzes** the response message to **distinguish** paired messages from unpaired messages and **stores** paired messages **and** unpaired messages in **separate** data structures. Consequently, Applicants submit that Claim 1-20 are in condition for allowance.

[0027] In the event any questions remain, the Examiner is respectfully requested to initiate a telephone conference with the undersigned.

Respectfully submitted,

Date: February 14, 2006

Kunzler & Associates
8 E. Broadway, Suite 600
Salt Lake City, Utah 84101
Telephone: 801/994-4646

David J. McKenzie
Reg. No. 46,919
Attorney for Applicant

Chapter 1. Introduction to OTMA

IMS Open Transaction Manager Access (OTMA) is a transaction-based, connectionless client/server protocol. Though easily generalized, its implementation is specific to IMS in a z/OS sysplex environment. The domain of the protocol is restricted to the domain of the z/OS Cross-System Coupling Facility (XCF).

OTMA addresses the problem of connecting a client to a server so that the client can support a large network, or a large number of sessions, while maintaining high performance.

Other solutions available today use network-based protocols, such as Systems Network Architecture (SNA). These protocols require a great amount of overhead because they are not transaction based.

The following topics provide additional information:

- "What is OTMA?"
- "How IMS Messages Flow in an OTMA Environment" on page 5
- "Using Transaction Pipes with OTMA" on page 8

What is OTMA?

OTMA has similarities to network protocols. There are several architectural models for networks. Figure 1 shows two. The simplified four-layer model shown on the right is often used in descriptions of UNIX® networks. In the open systems interconnection (OSI) model, shown on the left, OTMA is the session layer. Both models have a Transport, Network, and Data Link layer. The OSI model also includes layers for Application, Presentation, and Session, and the simplified model includes a process layer. In the four-layer model, OTMA is the process layer.

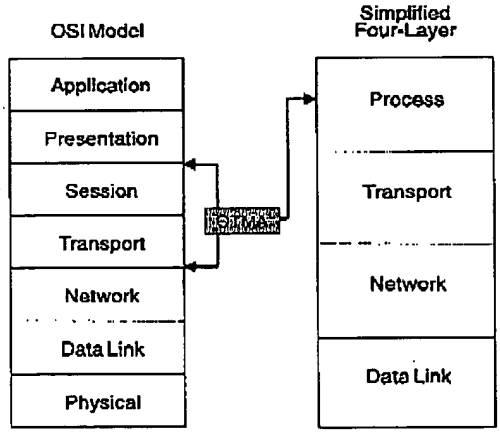

Figure 1. Network Architecture Models

OTMA, however, does not exactly conform to the OSI model, because OTMA can process several sessions simultaneously using a single transport connection, if the following are true:

- The z/OS Cross-System Coupling Facility (XCF) is the transport layer.

- A session is the connection between IMS and a client.
- A client or server only creates a single XCF connection.

OTMA performs some of the basic functions of the OSI transport layer (those not performed by XCF), so it is simplest to think of OTMA as a combined session and transport layer, with the transport layer comprised of both XCF and OTMA.

Although you can think of OTMA as a session and transport layer in a network architecture model, OTMA is designed to be a high-performance comprehensive protocol that allows z/OS programs to access IMS applications.

Definitions: A *z/OS program* in this case means any z/OS application that is a member of an XCF group that includes IMS. The XCF group members that IMS communicates with are called *OTMA clients*.

Related Reading: For more information on OTMA clients, see Chapter 2, "The OTMA Client," on page 13.

By using OTMA, each client (z/OS application) can submit transactions to IMS or issue IMS commands and receive output from IMS application programs and from IMS itself.

Definition: Because IMS can communicate with, or serve, many OTMA clients, IMS is called the *server*. However, OTMA only operates in the following IMS environments:

- IMS TM and DB (the IMS DB/DC environment)
- IMS TM with DB2™ UDB for z/OS (the IMS DCCTL environment)

Capabilities of OTMA

This section outlines the capabilities of OTMA

Existing IMS application programs can run without modification and interact with OTMA clients. APPC/IMS application programs that use IMS SETO calls might need some modification. **Related Reading:** For more information on this restriction, see "OTMA Restrictions" on page 50.

An OTMA client can issue most IMS commands and receive responses as a result of those commands. **Related Reading:** For information on the IMS commands that are supported, see the *IMS Version 9: Command Reference*.

An OTMA client can indicate that no security checking is to be done for its messages, thereby minimizing security-processing overhead.

The OTMA message flow and synchronization point protocols can be modified by an OTMA client for each transaction. In other words, the transaction-processing protocol used is not dependent on the current session.

The IMS `/DISPLAY TRANSACTION` command output is in the form of an OTMA message returned to the client in the application-data section of the message prefix.

OTMA-initiated transactions are identified to z/OS Workload Manager using the OTMA transaction-pipe name, which identifies the logical connection between IMS and OTMA.

Benefits of Using OTMA

This section outlines the benefits of OTMA.

Q: Do I need to modify my IMS applications?

A: No, but if you have applications that use SETO calls, you might have to modify them. The SETO call is relatively new and applies to APPC/IMS and SPOOL/API processing.

For each OTMA-originated transaction, the SETO call returns a status code. You can bracket the SETO call with an INQY call if necessary; see "Using DL/I Calls in an OTMA Environment" on page 55.

Full-duplex processing provides an environment in which transactions and output messages are sent and processed in parallel.

You can implement IMS device support outside IMS. You can also implement device support for your IMS subsystem that is different from what IMS provides, or enable device support that IMS does not provide. Figure 2 illustrates how IMS can communicate with a device, shown here as a workstation, using device support implemented within an OTMA client. IMS device support using Virtual Telecommunications Access Method (VTAM®) is shown for comparison.

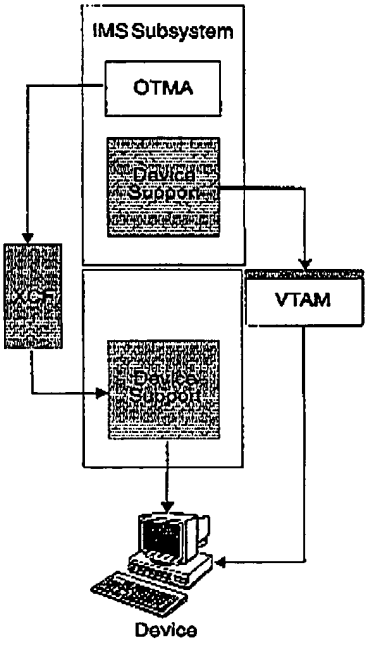

Figure 2. IMS communicates with a device using device support implemented within an OTMA Client

Flow-control and transaction-processing attributes are dynamically bound to the transaction.

What Is OTMA?

Clients have high-performance access to IMS:

- OTMA uses the z/OS XCF application programming interface (API).
- OTMA does not use VTAM and IMS device-dependent support.

Transactions based on different protocols (that is, that have different processing requirements such as being recoverable or irrecoverable) can be associated with a single transaction pipe. **Related Reading:** For more information on using transaction pipes, see "Using Transaction Pipes with OTMA" on page 8.

You can connect up to 255 clients to the OTMA group.

Messages can be extended using the user-data section of the message prefix, allowing additional user information to be sent with the transaction.

User information and transaction pipe name are included within the messages themselves.

Different clients can specify the same transaction pipe names, instead of needing to use uniquely named resources.

You do not need to use networking architectures, such as SNA (Systems Network Architecture).

Advantages of the OTMA Protocol

OTMA treats transactions as data objects that have attributes independent of application-, session-, or transport-layer considerations. OTMA is, in effect, a transaction layer, independent of other layers. As a unique layer, OTMA offers flexibility, simplicity, and performance that other solutions do not offer. This section outlines the transaction-specific services that OTMA provides the client.

Grouping of transactions using transaction pipes.

Security options (for example, the client can verify security or let the server verify the user ID).

Dynamically-bound flow control and processing. The client can decide how transaction requests and responses are to be processed by the server.

The ability for the client to query the server for transactions that the server supports.

Treating transactions as objects. The client can include any pertinent user data with the transaction, and allow that data to stay with all messages generated by the transaction.

The ability for the client to specify a client token with each transaction to correlate input with output.

The ability for the client to control transaction processing performed by the server, in terms of:

- Performance (the client can eliminate security-checking that the server performs).
- Transaction grouping, using the transaction-pipe token.

Client routing. An IMS exit routine can reroute an output message that is inserted to an alternate PCB to any OTMA client or to IMS.

Architected command output. The client can use the IMS /DISPLAY TRANSACTION command to query the server's transaction attributes and receive the reply in a structured format. Therefore, the need for automated operator scripting to control processing is reduced.

Unlike APPC, when using message flow through transaction pipes, no concept exists of a session that contains the flow-control parameters for all transactions and associated output data for the session.

How IMS Messages Flow in an OTMA Environment

Definition: The key to message flow for OTMA is the *transaction pipe*, the logical connection between the server and the OTMA client. An OTMA client includes the transaction-pipe name in the message-control information section of the message prefix for the input message. IMS then associates application output for an OTMA client with a specific transaction pipe.

Related Reading: For more information on transaction pipes, see "Using Transaction Pipes with OTMA" on page 8.

Basic OTMA Message Flow

The basic message flow is:

1. The client submits a transaction or command to IMS.
2. IMS accepts IMS transactions as input from any client.

   The IMS transaction code is specified in the application-data section of the input message.

   If the client is submitting an IMS command, the command is included in the application-data section of the input message.
3. The input message is processed.

   An IMS transaction is enqueued to the appropriate application program using an IMS scheduler message block (SMB).

   An IMS command is processed by IMS. The output is sent to the client synchronously or asynchronously, depending on the type of request.
4. Application output is sent to the client.

   Generation of output and commit are coordinated based on the commit mode specified in the state-data section of the message prefix for the input message.

   The application output is enqueued to a dynamically created IMS transaction-pipe structure (specific to that client) before being sent to the client.

   For an OTMA-submitted transaction, IOPCB output is returned to the OTMA client. By default, all alternate PCB output is also sent to the OTMA client. You can change this by coding the OTMA Prerouting exit routine (DFSYPRX0) or the client's OTMA Destination Resolution exit routine (DFSYDRU0). You can also use these exit routines to route alternate PCB output from non-OTMA-submitted transactions to OTMA clients.

   IMS delivers segmented messages in order, even though XCF does not guarantee sequential delivery of messages.

Figure 3 on page 6 shows an example of the message flow in an OTMA environment. Two clients are shown side by side in the example; they can be a

How IMS Messages Flow in an OTMA Environment

TCP/IP client, an WebSphere MQ Queue Manager client, or a client of any other network type. Message flow starts with the client, goes through the XCF group, and to IMS. Within the IMS address space, a control region contains OTMA; the message flow ends at a transaction-pipe. The IMS application program issues a Get Unique (GU) call in the dependent region.

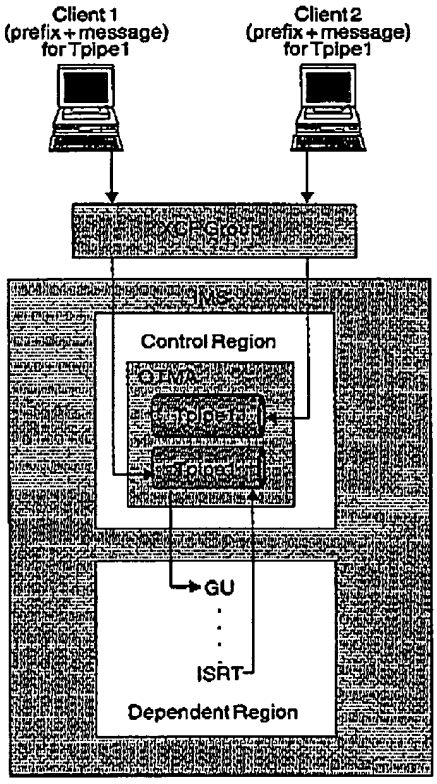

Figure 3. IMS Message Flow in an OTMA Environment

The notes in Figure 3 are as follows:

1. The message prefix is always attached to the input transaction, even in the case of segmented input. This prefix contains important information, such as the transaction-pipe name and the client token.

   A client application program can send several transactions specifying the same transaction-pipe name. The client token must always be present in the prefix, so that the client application program knows how to process the IMS output it receives.
2. OTMA clients do not need to predefine transaction pipes. Two different clients can use the same transaction-pipe name (as shown in Figure 3). Although many clients can use the same transaction-pipe name, each transaction pipe is unique. (In Figure 3, `client 1` and `client 2` both use `tpipe1`, yet each is a unique transaction pipe.)

   A client can create and use as many transaction pipes as it needs.
3. The transaction-pipe structure is created dynamically when OTMA receives output and is used as an anchor for the application output.

4. The IMS application program has no knowledge of the OTMA message prefix when it issues the GU call.

IMS supports a full-duplex message flow for a client/server session. The client can instead request a half-duplex message flow, but this flow must be implemented and managed by the client itself:

- A correlator token in the state-data section of the message prefix can be used to uniquely identify a transaction. IMS maintains this field in the message prefix for a transaction.
- The client can set the response-requested flag in the message-control information section of the message prefix to receive a response for a message.
- Any unsolicited output from IMS is easily identified by a client, because the message prefix specifies only the transaction-pipe name. The client can ask IMS to discard the output.

  Unsolicited output should not interfere with half-duplex processing. That is, the client must be prepared for full-duplex flows while still maintaining a half-duplex flow on a user-token level. Contention should not be an error condition.

Sample Commit-Then-Send Transaction Processing Flows

Figure 4 shows a non-OTMA environment: a secondary logical unit type 2 (SLU 2) device communicates with IMS using VTAM and IMS device support (DDMs). The transactions are enqueued to the IMS message queues. Transaction output is returned to the SLU 2 device.

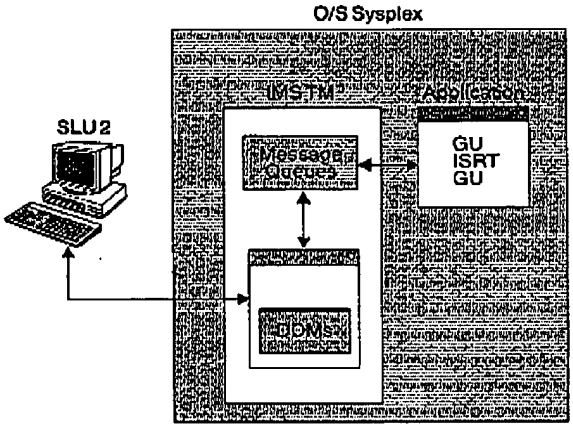

Figure 4. Standard SLU 2 Transaction Flow

Figure 5 on page 8 shows the same transaction flow in an OTMA environment. The transaction still comes from a SLU 2 device, but the device communicates with IMS using an OTMA client, through an XCF group, rather than VTAM.

Figure 5 on page 8 only shows the input flow, which begins with the SLU 2 device, goes to the OTMA client, through the XCF group, and ends at the OTMA server. The transaction is placed on the message queue, and the application issues get unique, insert, and get unique calls. Output follows the same path, in reverse. Of course, if a client is to send output to the SLU 2 device, the SLU 2 device must be defined to the client, and the client must be able to drive that device.

How IMS Messages Flow in an OTMA Environment

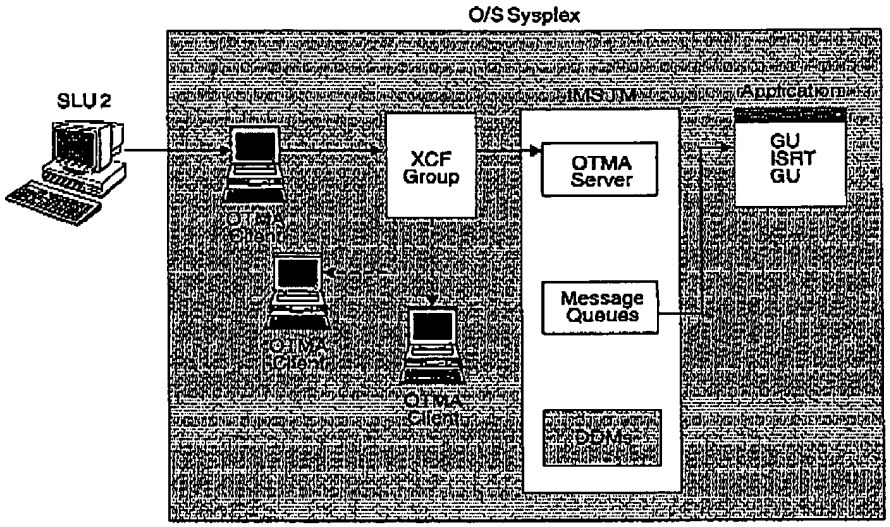

Figure 5. SLU 2 Transaction Flow Using OTMA

It might seem that the OTMA flow is more complex, and for a SLU 2 device, perhaps it is. But you can use OTMA to allow any type of device to communicate with IMS, not just VTAM-supported devices. An OTMA client can also act as a gateway for another network, such as a TCP/IP network.

Using Transaction Pipes with OTMA

An IMS transaction represents a request for IMS to do some work. Many transactions also require a response, after IMS has completed the work. So, each transaction has a source (the requester) and often a destination (for the response).

IMS uses the concept of a logical terminal (LTERM) to ensure that responses are associated with the correct requesters. An LTERM uses a queue where the transaction output is kept before it is returned to the requester.

Definition: For each LTERM, IMS maintains a connection between the queue and the physical node that receives the output. OTMA does not use an LTERM but still must maintain a connection between the client and IMS. This connection is the *transaction pipe*, or tpipe.

Q: What is a tpipe?

A: A transaction pipe (tpipe) is a logical connection between a client and the server. It is analogous to an IMS logical terminal (LTERM).

IMS uses the transaction-pipe name to associate all input and output with a particular client. The association between the transaction output and its ultimate destination (for example, a user at a terminal or a printer) is not made within IMS (as is the case with LTERMs), but is the responsibility of the client.

By using a transaction pipe, IMS does not know anything about the actual user of the transaction, often a user of the client application. Because IMS does not know anything about the actual user, the client has complete control over the output of transactions.

OTMA's use of transaction pipes provides:

- **Flexibility**

  Many transaction outputs can flow through the same transaction pipe.
- **Performance**

  Transaction pipes give the client the ability to specify and distinguish transactions based on their message-flow control and synchronization.
- **Resynchronization between a client and IMS**

  Transaction pipes can be either synchronized or non-synchronized. For a synchronized transaction pipe, all output messages are serialized through a single process, and sequence numbers can be assigned to messages. By logging these serialized messages, IMS and the client can resynchronize in the event of an outage.

  No resynchronization is required for a non-synchronized transaction pipe.
- **Object orientation**

  A transaction can be thought of as an object because OTMA keeps the transaction message information (such as user data and transaction-pipe name) within the message.

Figure 6 illustrates how transaction pipes fit in an OTMA client/server environment. As shown in Figure 6, transaction-pipe structures reside in the OTMA layer only for the server. XCF, which resides in the transport layer, can be thought of as an interprocess communication layer, because it provides communication between the client process and the server process.

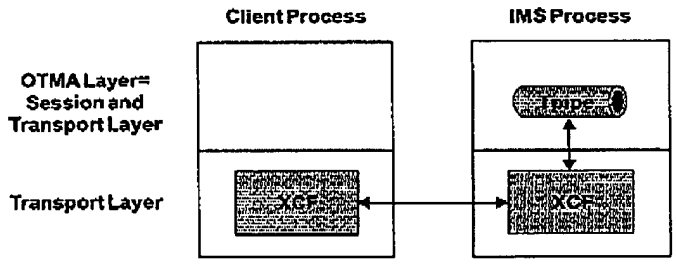

Figure 6. How Transaction Pipes Fit in an OTMA Client/Server Environment

Differences in Transaction Pipes

IMS LTERMs and UNIX pipes both provide a one-way flow for message traffic. An OTMA transaction pipe provides a two-way flow.

The concept of a transaction pipe is applicable to any protocol. In a general way, the transaction pipe replaces the IMS LTERM because:

- Processing is full duplex.
- Multiple flow-control mechanisms are possible.
- The logical output entity (in other words, the LTERM) is dissociated from the node of the actual user.

- The transaction pipe is implemented as a protocol rather than as an API, which facilitates a client/server architecture.
- The transaction pipe sets up a data-control mechanism independent of session characteristics, and is therefore transaction specific.

Message Flow Using Transaction Pipes

The flow control of transactions is handled by the client. The client dynamically binds flow-control parameters to the transaction by querying the transaction attributes in the server. Transaction pipes are not usually associated with flow control (except for synchronized transaction pipes using half-duplex processing).

Figure 7 shows the basic message flow between a client and a server, using XCF. The order of processing is:

1. The client sends a transaction as input to the server (IMS).
2. The server returns transaction output messages to the client.

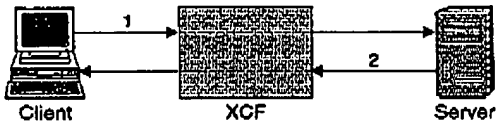

Figure 7. Basic Transaction-Pipe Message Flow

Within the server, the input transaction and the output messages are organized and synchronized using IMS queues, as shown in Figure 8 on page 11. The figure illustrates a commit-then-send transaction flow for a non-Fast Path environment.

The order of processing is:

1. The client sends a transaction to the server, and the server enqueues the transaction on a message queue.
2. The transaction is submitted to an application program for processing.
3. The application program prepares any output for the transaction and commits the output during sync-point processing.
4. The output is returned to the client.

Related Reading: For information on commit-then-send transactions, see "OTMA Commit Processing" on page 16.